Unleashing Hidden Powers of Inventor with the API

Part 4. Working with iProperties with Inventor API

Brian Ekins – Autodesk, Inc.

This last article in the series looks in detail at the iProperties portion of the programming interface and develops some practical examples.

Key Topics:

❑ The iProperties portion of the Inventor's object model

❑ iProperty program examples

Target Audience:

Autodesk Inventor users who want to increase their productivity with Inventor by writing programs. You should be familiar with VBA programming environment (See "Getting started with Inventor's VBA" article).

About the Author:
Brian is an Autodesk Inventor API evangelist and works with professionals and companies to help make using Inventor's programming interface easier. Brian began working in the CAD industry over 25 years ago. He has worked in CAD administration, as an application engineer, CAD API designer, and consultant. Brian was the original designer of Inventor's API and has presented at conferences and taught classes throughout the world to thousands of users and programmers.

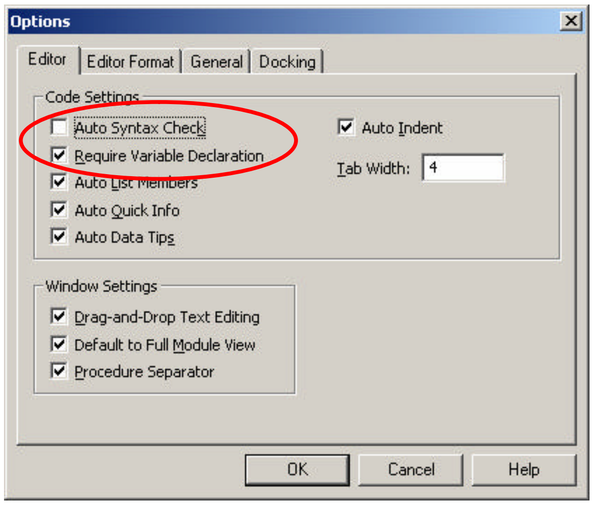
brian.ekins@autodesk.com

Introduction

The goal of this class is to provide enough information that anyone will be able to write a program to automate working with Inventor's iProperties. The article is not intended to be a coverage of VBA programming or even an exhaustive iProperties, but focuses on the features that are most commonly used. We'll begin by looking at the interface for programming iProperties. Finally, we'll look at several examples that demonstrate all of these concepts.

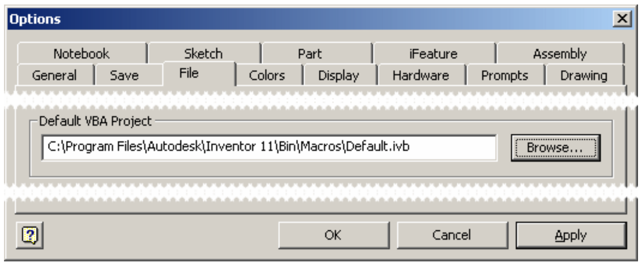
VBA Revisited

Visual Basic for Applications (VBA) is a programming environment developed by Microsoft and licensed by other companies to integrate into their applications. Autodesk licenses VBA and includes it in AutoCAD and Inventor.. This provides all customers of Inventor with a free development environment. You access VBA through Inventor using the **Macro | Visual Basic Editor** command in the Tools menu, or by pressing Alt-F11. Once the VBA environment is open, the first thing I recommend you do is change some of the VBA settings. In the VBA environment run the **Options** command from the Tools menu. Change the **Auto Syntax Check** and **Require Variable Declaration** settings to those shown below.



☐ Auto Syntax Check
☑ Require Variable Declaration
☑ Auto List Members
☑ Auto Quick Info
☑ Auto Data Tips
☑ Auto Indent
Tab Width: 4
☑ Drag-and-Drop Text Editing
☑ Default to Full Module View
☑ Procedure Separator

The VBA programs you write can be saved in Inventor documents or in external files. For 99% of the cases, saving it in an external file is best, so that's what we'll cover here. This external file is referred to as a VBA project and Inventor saves it as an .ivb file. A single project can contain any number of macros (programs). There is a default VBA project that Inventor loads at startup. The default VBA project is defined in the File tab of the Inventor application options, as shown below. This is where your programs will be saved.



Default VBA Project: C:\Program Files\Autodesk\Inventor 11\Bin\Macros\Default.ivb

Apprentice

There's one other technique of accessing some of Inventor's functionality. There's a utility component referred to as *Apprentice* that provides a programming interface to a small portion of the information in an Inventor document. Apprentice doesn't have a user-interface and is exclusively a programming component. Design Assistant and Inventor View are both programs that use Apprentice to access document information. iProperties are one of the things that you have access to through Apprentice.

There are a couple of reasons to use Apprentice when possible. First, programs using Apprentice will run much faster than the equivalent program in Inventor. This is because Apprentice doesn't have a user-interface and it only provides partial access to information in the document so it doesn't need to load as much of the document as Inventor does. Second, Apprentice is freely available by installing Inventor View, (which includes Apprentice), from autodesk.com.

We'll see some examples of the use of apprentice in some of the samples.

Collection Objects

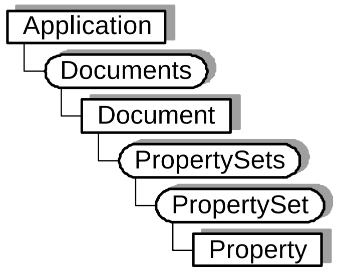
Another concept that's important to understand when working with Inventor's programming interface is *collection* objects. Collection objects are represented in the object model diagram by boxes with rounded corners. In the object model picture to the right the Documents, PropertySets, and PropertySet objects are collection objects.

The primary job of a collection object is to provide access to a group of related objects (the set of children for that parent). For example, the Documents object provides access to all of the documents that are currently open in Inventor. A collection provides access to its contents through two properties; *Count* and *Item*. The Count property tells you how many items are in the collection and the Item property returns a specific item. All collections support these two properties. Another important feature of many collections is the ability to create new objects. For example the Documents collection supports the Add method which is used to create new documents. It also supports the Open method which is used to open existing documents from the disk. These will be used in some of the samples that follow.

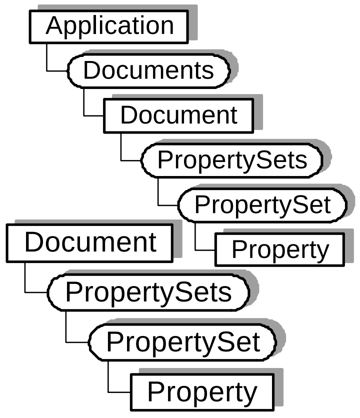
Programming Inventor's iProperties

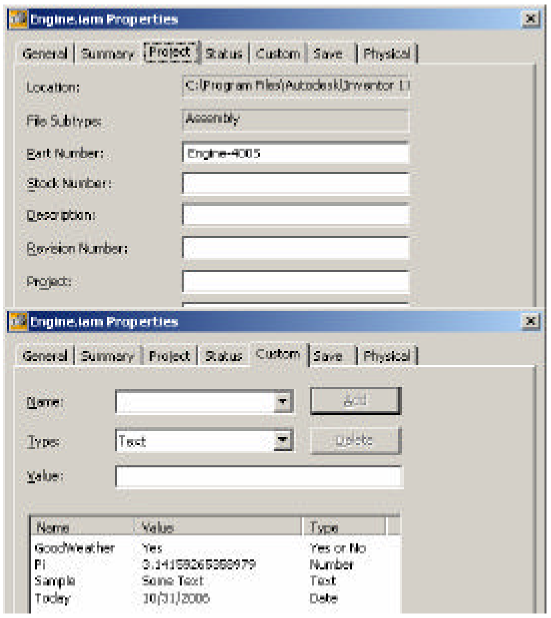
iProperties in the User-Interface

Before looking at the programming interface for iProperties let's do a quick review of iProperties from an end-user perspective. iProperties are accessed through the iProperties dialog. There are several tabs at the top that organize the available properties. There are also some tabs on this dialog that are not actually iProperties. For example, the General tab provides access to information about the document, the Save tab provides access to options that control how the thumbnail image is captured, and the Physical tab provides access to the various physical properties of the part or assembly document. If you need access to this information it is available through programming objects other than properties.



The picture below on the left illustrates a typical tab of the iProperties dialog where you have access to a specific set of iProperties. Through the dialog, you can view and edit the values of iProperties. The set of iProperties shown on each tab are predetermined by Inventor and cannot be changed. However, using the Custom tab (shown in the picture below on the right) you can add additional iProperties to the document. This allows you to associate any information you want with the document.





Engine.iam Properties — Project

Location:	C:\Program Files\Autodesk\Inventor 1
File Subtype:	Assembly
Part Number:	Engine-4005
Stock Number:	
Description:	
Revision Number:	
Project:	

Engine.iam Properties — Custom

Name:	
Type:	Text
Value:	

Name	Value	Type
GoodWeather	Yes	Yes or No
Pi	3.14159265358979	Number
Sample	Some Text	Text
Today	10/31/2006	Date

iProperties in Inventor's Programming Interface

The object model for iProperties is shown to the right. The diagram starts with the Application object and goes through the Documents collection to get the Document object. Remember that the Document object is the base class for all document types so it can represent any type of Inventor document. From the Document object we then access the PropertySets object which is the top-level property related object and provides access to the various iProperties associated with that particular document.

Accessing iProperites

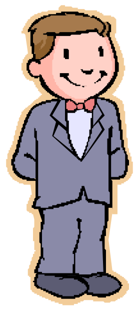
Accessing iProperties through Inventor's programming interface is reasonably simple. The object model for iProperties is shown to the right. The first thing to notice is that they're accessed through the Document object. Properties are owned by documents and to get access to properties you need to go through the document that owns them.

Even though the programming interface for iProperties is simple, people still tend to struggle with it. I believe this is primarily because of not understanding how to access a specific property. Before discussing iProperties in detail, a brief discussion of naming is appropriate to help describe the concepts Inventor uses. The picture below shows a person and three different ways to identify this person. His full legal name is a good way to identify him, although a bit formal. His social security number is good, but not very user friendly. His nickname, although commonly used, can have problems since it's not as likely to be unique Bill could decide tomorrow he would rather be called Billy or Will. The point is that there are three ways to identify this person, each one with its own pros and cons.

Legal Name: William Harry Potter

Nickname: Bill

SSN: 365-58-9401



Legal Name: William Harry Potter

Nickname: Bill

SSN: 365-58-9401

When working with iProperties you'll also need to specify specific objects.   Like Bill, iProperties also have several names. Understanding this single concept should help you overcome most of the problems other people have had when working with iProperties. The various iProperty objects, their names and best use suggestions are described below.

PropertySets Objects

The PropertySets object serves as the access point to iProperties. The PropertySets object itself doesn't have a name but is simply a utility collection object that provides access to the various PropertySet objects. Using the Item method of the PropertySets object you'll specify which PropertySet object you want. The Item method accepts an Integer value indicating the index of the PropertySet object you want, but more important, it also accepts the name of the PropertySet object you want. The next section discusses PropertySet objects and their various names.

PropertySet Objects

The PropertySet object is a collection object and provides access to a set of iProperties. The PropertySet object is roughly equivalent to a tab on the iProperties dialog.   The Summary, Project, Status, and Custom tabs of the dialog contain the iProperties that are exposed through the programming interface. There are four PropertySet objects in an Inventor document; Summary Information, Document Summary Information, Design Tracking Properties, and User Defined Properties. (You will find the details of these property sets and the pre-defined properties at the end of this article).

PropertySet objects are named so that you can find a particular PropertySet object. A PropertySet object has three names; Name, Internal Name, and Display Name. Using the analogy of Bill above, the Name is equivalent to his legal name, the Internal Name is equivalent to his social security number, and the Display Name is equivalent to his nickname. Let's look at a specific example to illustrate this. There is a PropertySet object that has the following names:

Name: Inventor Summary Information
Internal Name: {F29F85E0-4FF9-1068-AB91-08002B27B3D9}
DisplayName: Summary Information

Any one of these can be used as input to the Item method in order to get this particular PropertySet object. I would suggest always using the Name for the following reasons. The Name cannot be changed, is guaranteed to be consistent over time, and is an English human readable string. The Internal Name cannot be changed and will remain consistent but it is not very user-friendly and makes your source code more difficult to read and maintain. The Display Name is not guaranteed to remain constant. The Display Name is the localized version of the name and will change for each language. A chart showing the names of the four standard PropertySet objects is at the end of this paper.

Below is an example of obtaining one of the PropertySet objects. In this case the summary information set of iProperties.

```
Public Sub GetPropertySetSample()
    ' Get the active document.
    Dim invDoc As Document
    Set invDoc = ThisApplication.ActiveDocument

    ' Get the summary information property set.
    Dim invSummaryInfo As PropertySet
    Set invSummaryInfo = invDoc.PropertySets.Item("Inventor Summary Information")
End Sub
```

Property Objects

A Property object represents an individual property. Each Property object also has three names; Name, Display Name, and ID. Many of the same principles discussed for PropertySet objects applies here. The Name is an English string that is guaranteed to remain constant. The Display Name is the localized version of the Name and can change, so it's not a reliable method of accessing a particular property. The ID is a number and is similar to the Internal Name of the PropertySet object, but is a simple Integer number instead of a GUID. For consistency I would recommend using the name of the property when you need to access a specific one. Below is an example of the three identifiers for a particular property.

Name: Part Number
DisplayName: Part Number
ID: 5 or kPartNumberDesignTrackingProperties

The following code gets the iProperty that represents the part number.

```
Public Sub GetPropertySample()
    ' Get the active document.
    Dim invDoc As Document
    Set invDoc = ThisApplication.ActiveDocument

    ' Get the design tracking property set.
    Dim invDesignInfo As PropertySet
    Set invDesignInfo = invDoc.PropertySets.Item("Design Tracking Properties")

    ' Get the part number property.
    Dim invPartNumberProperty As Property
    Set invPartNumberProperty = invDesignInfo.Item("Part Number")
End Sub
```

You may see program samples that use identifiers like kPartNumberDesignTrackingProperties to specify a specific property. These identifiers are defined in the Inventor type library and provide a convenient way of specifying the ID of a property. For the part number, instead of specifying the ID as 5 you can use kPartNumberDesignTrackingProperties. This makes your code more readable. If you want to use the ID instead of the Name you need to use the ItemByPropId property instead of the standard Item property of the PropertySets object. As stated before, for consistency I would recommend using the Name in both cases.

Something else you'll see in a lot of sample code is where several property calls are combined into a single line. The code below does the same thing as the previous sample but it's getting the PropertySet object returned by the PropertySets Item property and immediately calling the Item property on it to get the desired property.

```
Public Sub GetPropertySample()
    ' Get the active document.
    Dim invDoc As Document
    Set invDoc = ThisApplication.ActiveDocument

    ' Get the part number property.
    Dim invPartNumberProperty As Property
    Set invPartNumberProperty = invDoc. _
            PropertySets.Item("Design Tracking Properties").Item("Part Number")
End Sub
```

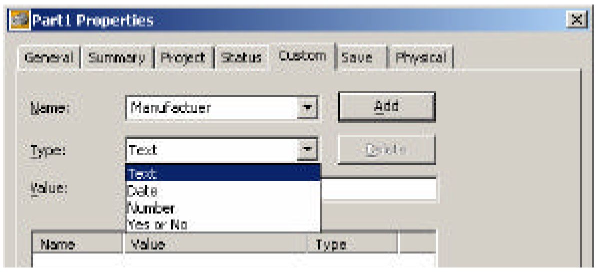
Now that we have a reference to a specific property we can use its programming properties to get and set the value. The Property object supports a property called Value that provides this capability. For example, the line below can be added to the previous sample to display the current part number.

```
MsgBox "The part number is: " & invPartNumberProperty.Value
```

This next line sets the value of the part number iProperty.

```
invPartNumberProperty.Value = "Part-001"
```

The Value property uses an interesting programming feature that is useful to understand when working with iProperties. The Value property is of type *Variant*. A Variant is a special type that can represent almost any other type. For example, a Variant can hold a String, Double, Date, or most any other type. iProperties take advantage of this since they allow the value to be one of many different types. You see this when working with custom (user defined) properties, as shown in the picture below. When creating or modifying a custom iProperty you define the type; Text, Date, Number, or Yes or No. This results in an iProperty being created where the value contains a String, Date, Double, or Boolean value.



Name: Manufactuer

Type: Text

Value:

Text
Date
Number
Yes or No

When setting the value of any of Inventor's predefined iProperties, Inventor forces them to be the correct type and will convert them automatically, when possible. If you supply a value that can't be converted to the expected type, it will fail. In the table at the end of this paper the type of each property is listed. Most of the standard iProperties are Strings, with a few Date, Currency, Boolean, and Long values. There is also one other slightly unusual type called IPictureDisp. This type is used for the thumbnail picture associated with a document. Using this you can extract the thumbnail picture from a document.

Creating Properties

The primary reason to be aware of the Variant type is when you create your own custom iProperties. Interactively, as shown in the previous picture, you specify the type of property you're going to create; Text, Date, Number, or Yes or No. When you create them using Inventor's programming interface you don't explicitly specify the type but it is implicitly determined based on the type of variable you input.

New iPropertiies can only be created within the Custom (user defined) set of properties. New iProperties are created using the Add method of the PropertySet object. The sample below illustrates creating four new iProperties, one of each type.

```
Public Sub CreateCustomProperties()
    ' Get the active document.
    Dim invDoc As Document
    Set invDoc = ThisApplication.ActiveDocument

    ' Get the user defined (custom) property set.
    Dim invCustomPropertySet As PropertySet
    Set invCustomPropertySet = invDoc.PropertySets.Item( _
                                        "Inventor User Defined Properties")

    ' Declare some variables that will contain the various values.
    Dim strText As String
    Dim dblValue As Double
    Dim dtDate As Date
    Dim blYesOrNo As Boolean

    ' Set values for the variables.
    strText = "Some sample text."
    dblValue = 3.14159
    dtDate = Now
    blYesOrNo = True

    ' Create the properties.
    Dim invProperty As Property
    Set invProperty = invCustomPropertySet.Add(strText, "Test Test")
    Set invProperty = invCustomPropertySet.Add(dblValue, "Test Value")
    Set invProperty = invCustomPropertySet.Add(dtDate, "Test Date")
    Set invProperty = invCustomPropertySet.Add(blYesOrNo, "Test Yes or No")
End Sub
```

A common task is when you have a value you want to save as a custom property within a document. If the property already exists you just want to update the value. If the property doesn't exist then you need to create it with the correct value. The code below demonstrates getting the volume of a part and writing it to a custom property named "Volume". With the volume as an iProperty it can be used as input for text on a drawing. The portion of this macro that gets the volume and formats the result is outside the scope of this paper but helps to demonstrate a practical use of creating and setting the value of an iProperty.

```
Public Sub UpdateVolume()
    ' Get the active part document.
    Dim invPartDoc As PartDocument
    Set invPartDoc = ThisApplication.ActiveDocument

    ' Get the volume of the part.  This will be returned in
    ' cubic centimeters.
    Dim dVolume As Double
    dVolume = invPartDoc.ComponentDefinition.MassProperties.Volume

    ' Get the UnitsOfMeasure object which is used to do unit conversions.
    Dim oUOM As UnitsOfMeasure
    Set oUOM = invPartDoc.UnitsOfMeasure

    ' Convert the volume to the current document units.
    Dim strVolume As String
    strVolume = oUOM.GetStringFromValue(dVolume, _
                               oUOM.GetStringFromType(oUOM.LengthUnits) & "^3")

    ' Get the custom property set.
    Dim invCustomPropertySet As PropertySet
    Set invCustomPropertySet = _
                  invPartDoc.PropertySets.Item("Inventor User Defined Properties")

    ' Attempt to get an existing custom property named "Volume".
    On Error Resume Next
    Dim invVolumeProperty As Property
    Set invVolumeProperty = invCustomPropertySet.Item("Volume")
    If Err.Number <> 0 Then
        ' Failed to get the property, which means it doesn't exist
        ' so we'll create it.
        Call invCustomPropertySet.Add(strVolume, "Volume")
    Else
        ' We got the property so update the value.
        invVolumeProperty.Value = strVolume
    End If
End Sub
```

Saving Properties

To save any changes you make to properties you need to save the document. The Save method of the Document object does this. However, when working with Apprentice it's possible to only save the iProperty changes to the document, which is much faster than saving the entire document. The FlushToFile method of the PropertySets object saves any iProperty changes. The sample below demonstrates this by opening a document using Apprentice, changing a property value, saving the change, and closing everything. Remember that Apprentice cannot be used from within Inventor's VBA. It must be from another application's VBA or from Visual Basic.

```
Public Sub ApprenticeUpdate()
    ' Declare a variable for Apprentice.
    Dim invApprentice As New ApprenticeServerComponent

    ' Open a document using Apprentice.
    Dim invDoc As ApprenticeServerDocument
    Set invDoc = invApprentice.Open("C:\Temp\Part1.ipt")

    ' Get the design tracking property set.
    Dim invDTProperties As PropertySet
    Set invDTProperties = invDoc.PropertySets.Item("Design Tracking Properties")

    ' Edit the values of a couple of properties.
    invDTProperties.Item("Checked By").Value = "Bob"
    invDTProperties.Item("Date Checked").Value = Now

    ' Save the changes.
    invDoc.PropertySets.FlushToFile
    ' Close the document and release all references.
    Set invDoc = Nothing
    invApprentice.Close
    Set invApprentice = Nothing
End Sub
```

Example Programs

There are some associated sample programs that provide more complete examples of the various topics covered in this paper and provide practical examples in various programming languages of how to use Inventor's programming interface.

PropertySamples.ivb

CopyProperties – Copies a set of properties from a selected document into the active document.

DumpPropertyInfo – Displays information about all of the property sets and their properties.

Properties.xls

An Excel file where each sheet contains a list of properties and values. There are two Excel macros that are executed using the buttons on the first sheet.

Set Properties of Active Document – Copies the properties of the selected Excel sheet into the active document.

Set Properties of Documents – Copies the properties of the selected Excel sheet into all of the Inventor documents in the selected directory. This sample uses Apprentice.

You can find these files accompanied with this article.

Inventor User Defined Properties, {D5CDD505-2E9C-101B-9397-08002B2CF9AE}			
Inventor Summary Information, {F29F85E0-4FF9-1068-AB91-08002B27B3D9}			
Property Name	**Id**	**Id Enum**	**Type**
Title	2	kTitleSummaryInformation	String
Subject	3	kSubjectSummaryInformation	String
Author	4	kAuthorSummaryInformation	String
Keywords	5	kKeywordsSummaryInformation	String
Comments	6	kCommentsSummaryInformation	String
Last Saved By	8	kLastSavedBySummaryInformation	String
Revision Number	9	kRevisionSummaryInformation	String
Thumbnail	17	kThumbnailSummaryInformation	IPictureDisp
Inventor Document Summary Information, {D5CDD502-2E9C-101B-9397-08002B2CF9AE}			
Property Name	**Id**	**Id Enum**	**Type**
Category	2	kCategoryDocSummaryInformation	String
Manager	14	kManagerDocSummaryInformation	String
Company	15	kCompanyDocSummaryInformation	String

Design Tracking Properties, {32853F0F-3444-11D1-9E93-0060B03C1CA6}			
Property Name	**Id**	**Id Enum**	**Type**
Creation Time	4	kCreationDateDesignTrackingProperties	Date
Part Number	5	kPartNumberDesignTrackingProperties	String
Project	7	kProjectDesignTrackingProperties	String
Cost Center	9	kCostCenterDesignTrackingProperties	String
Checked By	10	kCheckedByDesignTrackingProperties	String
Date Checked	11	kDateCheckedDesignTrackingProperties	Date
Engr Approved By	12	kEngrApprovedByDesignTrackingProperties	String
Engr Date Approved	13	kDateEngrApprovedDesignTrackingProperties	Date
User Status	17	kUserStatusDesignTrackingProperties	String
Material	20	kMaterialDesignTrackingProperties	String
Part Property Revision Id	21	kPartPropRevIdDesignTrackingProperties	String
Catalog Web Link	23	kCatalogWebLinkDesignTrackingProperties	String
Part Icon	28	kPartIconDesignTrackingProperties	IPictureDisp
Description	29	kDescriptionDesignTrackingProperties	String
Vendor	30	kVendorDesignTrackingProperties	String
Document SubType	31	kDocSubTypeDesignTrackingProperties	String
Document SubType Name	32	kDocSubTypeNameDesignTrackingProperties	String
Proxy Refresh Date	33	kProxyRefreshDateDesignTrackingProperties	Date
Mfg Approved By	34	kMfgApprovedByDesignTrackingProperties	String
Mfg Date Approved	35	kDateMfgApprovedDesignTrackingProperties	Date
Cost	36	kCostDesignTrackingProperties	Currency
Standard	37	kStandardDesignTrackingProperties	String
Design Status	40	kDesignStatusDesignTrackingProperties	Long
Designer	41	kDesignerDesignTrackingProperties	String
Engineer	42	kEngineerDesignTrackingProperties	String
Authority	43	kAuthorityDesignTrackingProperties	String
Parameterized Template	44	kParameterizedTemplateDesignTrackingProperties	Boolean
Template Row	45	kTemplateRowDesignTrackingProperties	String
External Property Revision Id	46	kExternalPropRevIdDesignTrackingProperties	String
Standard Revision	47	kStandardRevisionDesignTrackingProperties	String
Manufacturer	48	kManufacturerDesignTrackingProperties	String
Standards Organization	49	kStandardsOrganizationDesignTrackingProperties	String
Language	50	kLanguageDesignTrackingProperties	String
Defer Updates	51	kDrawingDeferUpdateDesignTrackingProperties	Boolean
Size Designation	52		String
Categories	56	kCategoriesDesignTrackingProperties	String
Stock Number	55	kStockNumberDesignTrackingProperties	String
Weld Material	57	kWeldMaterialDesignTrackingProperties	String